Technischer Überblick zu Web Services

Stephan Karrer

Frühere Architektur-Ansätze für verteilte Applikationen und Infrastruktur:

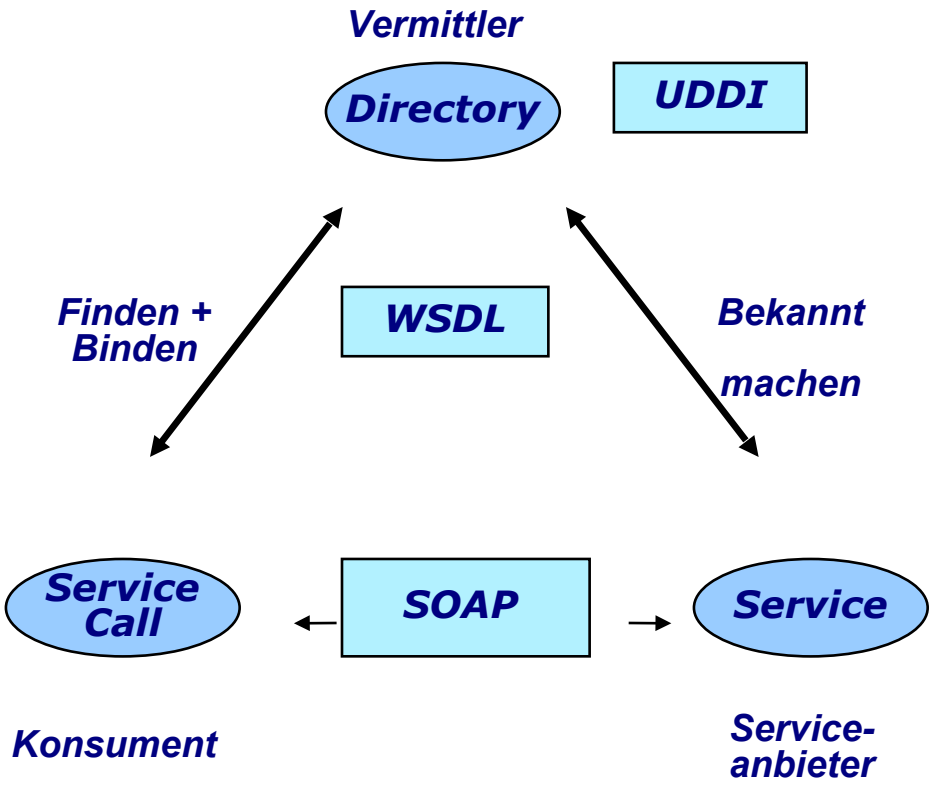
- Proprietär, z.B. Datenbankhersteller wie Oracle, IBM, ...
- RPC-Konzepte: Sun RPC (UNIX-Welt), DCE-RPC (Microsoft)
- CORBA (OMG)
- RMI (Java)

  – aber letztendlich bis heute nicht vollständig Plattform und Hersteller-übergreifend

  Die zu lösenden technischen Herausforderungen sind nach wie vor gegeben:

  Web Services als kleinster gemeinsamer Nenner ?

Web Services: Basistechnologien



Web Services sind keine Revolution sondern eher Evolution durch Kombination verschiedener Technologien:

- HTTP, SHTTP, SMTP
- XML

und Konzepte

Auf dieser Basis entstehen:

SOAP, WSDL, UDDI, ...

HTTP als Transport-Protokoll

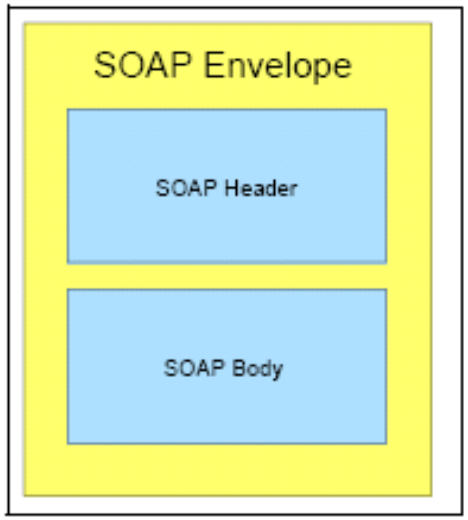
- **Vorteile:**
  - weitreichende Unterstützung und Verfügbarkeit
  - offenes Protokoll
  - HTTP-Daten passieren in der Regel vorhandene Firewalls
  - sichere Kommunikation kann via HTTPS erfolgen

- **Nachteile:**
  - HTTP bietet keine spezielle Servicequalität (QoS)
  - HTTP ist „stateless“
    - ☞ hier muß bei Bedarf auf der Applikationsebene nachgesteuert werden
    - ☞ man kann auch andere Transport-Protokolle verwenden

Weitere Transport-Protokolle (SOAP bindings)

- SOAP Over Email
  - im Body der Email
  - als Anhang (Verwendung von MIME)
- Direkte Verwendung des Transport-Protokolls
  - z.B. UDP, TCP, …
- SOAP über JMS
  - z.B. Nutzung von Websphere MQ

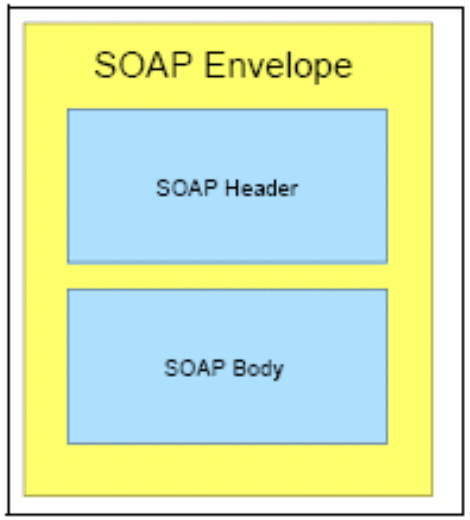
SOAP Nachricht (Simple Object Access Protocol)



- SOAP spezifiziert Nachrichten im XML-Format
- SOAP definiert einen Rahmen für die Inhalte der Nachricht und Regeln, wie diese zu verarbeiten sind
- SOAP definiert auch Regeln für die Bindung an darunterliegende Transport-Protokolle
- Es existieren Codier/Decodier-Schemata für grundlegende Datentypen („Encoded“, was aber nicht aktuell ist) oder es wird XSD benutzt

```
<?xml version="1.0">
<env:Envelope xmlns:env = http://www.w3c.org/soap-envelope>
        <env:Header> … </env:Header>*
        <env:Body>
                …
                <env:Fault> … </env:Fault>*
                …
        </env:Body>
</env:Envelope>
```

SOAP Nachricht: Bestandteile



- SOAP definiert nicht den Inhalt der einzelnen Elemente wie Header und Body
- Der optionale Header enthält Kontroll-Informationen aus der Anwendungssicht:
  - Welche Knoten müssen was bearbeiten
  - Zustellhinweise
  - Kontext-Informationen bzgl. des Inhalts
- Der Body enthält die eigentliche, applikationsspezifische Nachricht
- Im Falle einer Fehler-Signalisierung besteht der Body aus den Fehlerinformationen

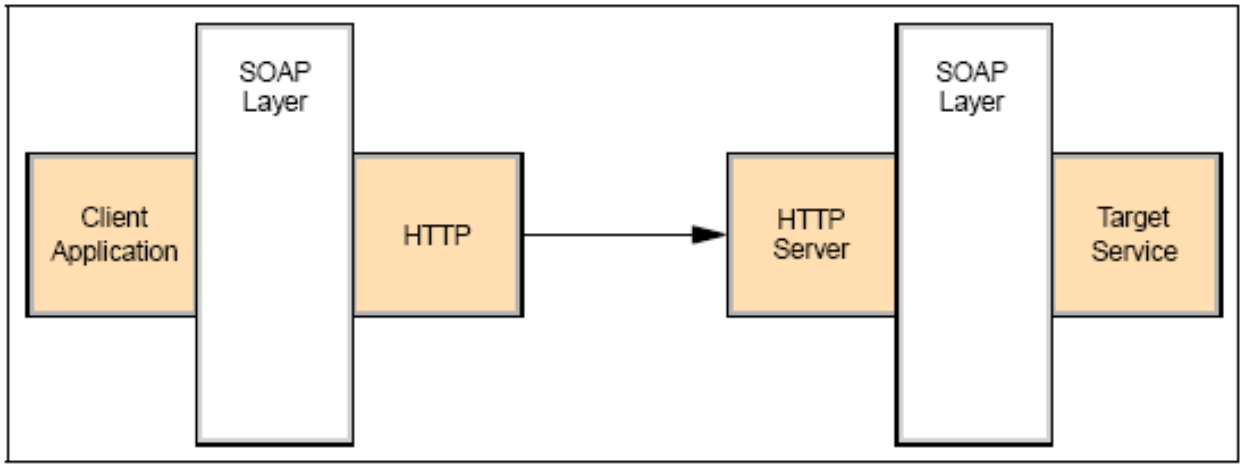
SOAP Document Beispiel:

Reisereservierung

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
         env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
          env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
</env:Header>
 <env:Body>
  <p:departure xmlns:p="http://travelcompany.example.org/reservation/travel">
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2001-12-14</p:departureDate>
  </p:departure>
  <q:lodging
   xmlns:q="http://travelcompany.example.org/reservation/hotels">
   <q:preference>none</q:preference>
  </q:lodging>
 </env:Body>
</env:Envelope>
```

Quelle: W3C

Verarbeitung eines Service-Calls



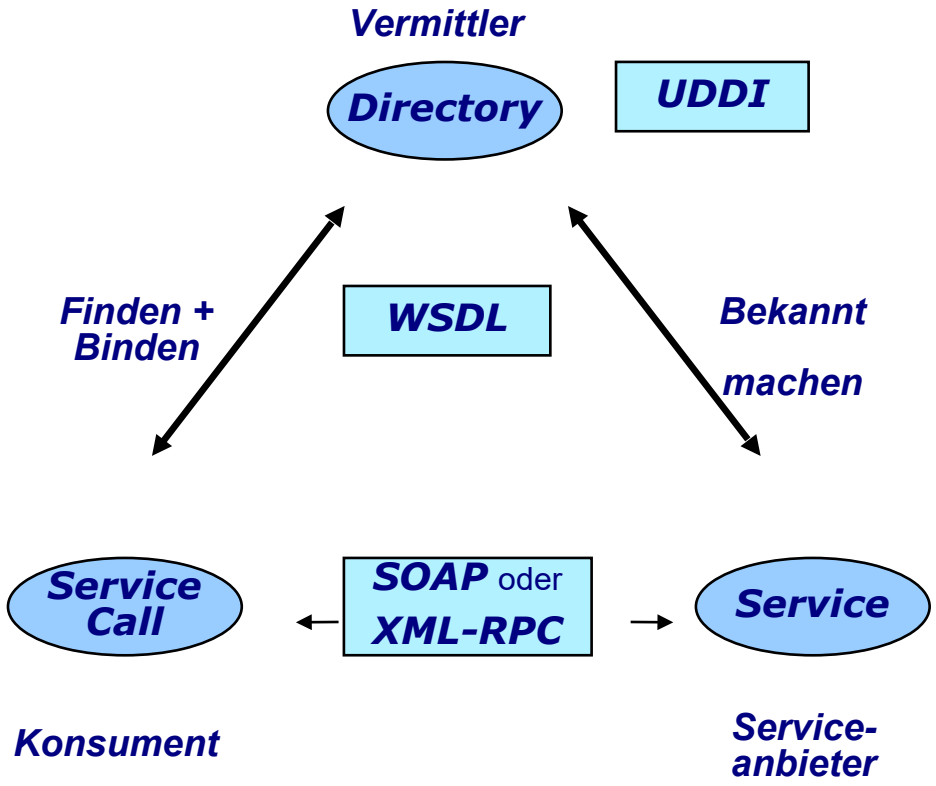
1. Client ruft Methode (Prozedur) via Proxy (Stub) auf
2. SOAP-Layer wandelt den Aufruf in XML-Format
3. SOAP-Layer packt das XML-Format in ein SOAP-Paket (ebenfalls XML-Format)
4. SOAP-Nachricht wird an URI verschickt

5. Web-Server empfängt SOAP-Nachricht
6. SOAP-Layer extrahiert den XML-Aufruf aus der SOAP-Nachricht
7. SOAP-Layer wandelt das XML-Format in einen Aufruf für die Zielumgebung
8. Der Service wird lokalisiert und aufgerufen

Generierung des SOAP-Layers

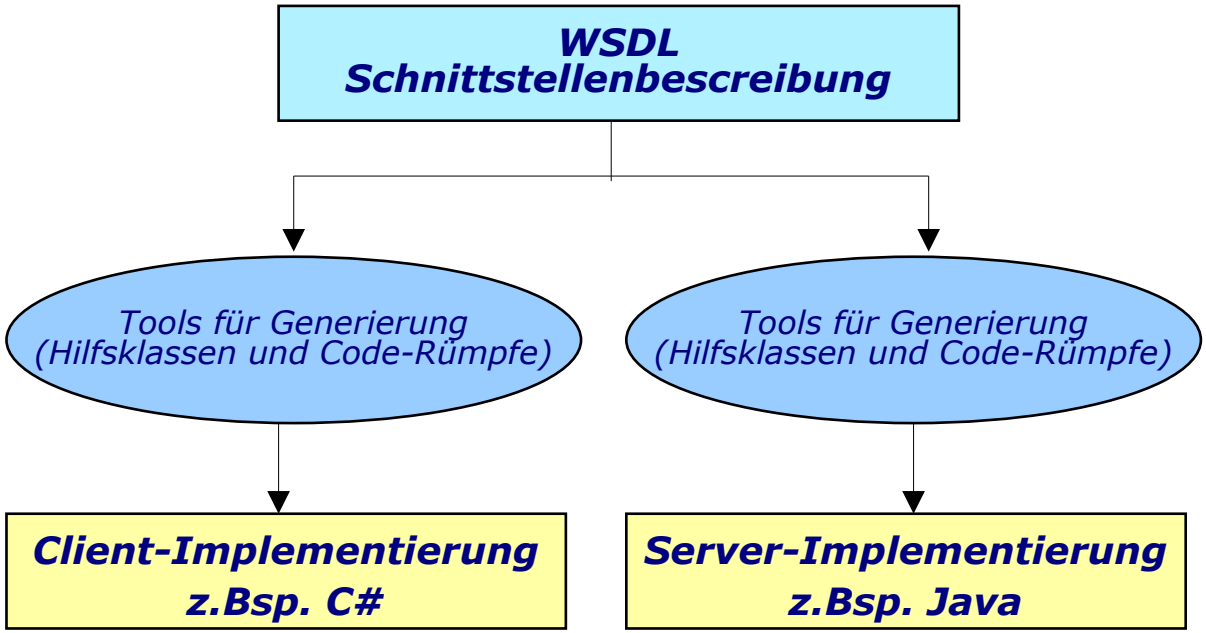
- Die Aufgaben des SOAP-Layers auf der Client- sowie Server-Seite können in hohem Maße automatisiert werden.
- Es existieren für viele Zielsprachen, Laufzeitumgebungen und Transportmechanismen entsprechende Werkzeuge für die Entwickler
- Häufig sind die Services schon vorhanden:
  Entsprechende Werkzeuge generieren dann anhand des vorhandenen Codes möglichst komplett den SOAP-Layer
  - Die Implementierung ist damit üblicherweise von dem Netzwerk- und SOAP-Layer entkoppelt

WSDL (Web Service Description Language)



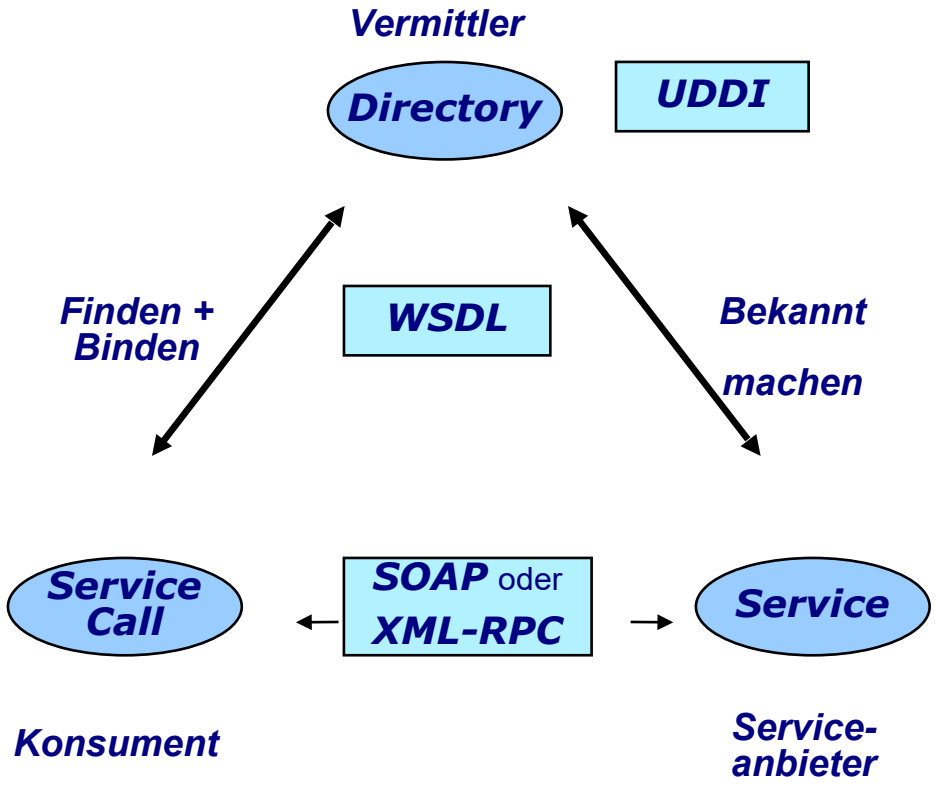
- Wie bei früheren Konzepten kommt auch hier der Schnittstelle zwischen Aufrufer und Aufgerufenem überragende Bedeutung zu
- Der Service wird mittels eines XML-Dialekts, der WSDL, beschrieben
- WSDL entspricht den IDLs anderer Konzepte
- Damit ist quasi die Selbstbeschreibung gegeben (self description)
- Die neueste Version ist WSDL Version 2.0 (W3C Recommendation 26 June 2007), am Markt etabliert ist aber WSDL Version 1.1

WSDL (Web Service Description Language)



- Anhand der Service-Beschreibung im XML-Format werden üblicherweise die erforderlichen Artefakte für das jeweilige WebService-Framework generiert.
- Das WebService-Framework übernimmt die Abbildung von/zu XML und den SOAP- und Netzwerk-Layer.

Was ist UDDI



- Universal Description, Discovery, and Integration
- UDDI spezifiziert Protokole für:
  - Veröffentlichung von Services in Verzeichnissen
  - Suchmöglichkeiten nach Services in Verzeichnissen
  - Zugriffskontrolle auf Verzeichnisse
  - Verteilung auf verschiedene Verzeichnisse
- Bietet Lokations- und Aufruf-Metadaten für die dynamische Bindung von Konsumenten an Services zur Laufzeit
- Wird durch das OASIS Standardisierungs-Gremium verwaltet
- Im Java-Umfeld spielt der entsprechende Standard JAXR kaum eine Rolle

Grenzen der Spezifikation

- **Verschiedene Styles:**
  - RPC/Literal
  - RPC/Encoded
  - Document/Literal
  - Document/Encoded
  - Signalisierung der Operation via HTTp-Header (SOAPAction) oder nicht

    ➢ OASIS (Industrie-Konsortium) definiert deshalb Basic Profile für “Web Services Interoperability“

- **Ein ganzer Zoo von optionalen WS-*-Erweiterungen**
  - WS Security
  - WS Adressing
  - ...

- **HTTP als Transport-Protokoll:**
  - HTTP bietet keine spezielle Servicequalität (QoS)
  - HTTP ist „stateless“

    ☞ hier muß bei Bedarf auf der Applikationsebene nachgesteuert werden

    ☞ man kann auch andere Transport-Protokolle verwenden

Der Standard JAX-WS

- JAX-RPC (Java API for XML-based RPC) war der Vorläufer
  - Apache Axis (Weiterentwicklung Apache SOAP, ursprüngliche Basis „SOAP4J“ von IBM) war die Referenz-Implementierung
  - heute üblicherweise nicht mehr im Einsatz, entsprechende APIs sind deprecated
- JAX-WS Implementierungen
  - Eclipse Metro ist die Referenz-Implementierung
  - Apache CXF
  - Apache Axis 2
- JAX-WS ist Teil der JEE- bzw. Jakarta EE-Spezifikation 8
  - Somit Bestandteil eines jeden JEE8 -kompatiblen Applikationsserver
  - Jakarta EE-Spezifikation 9 macht JAX-WS zum optionalen Bestandteil !!
- JAX-WS setzt auf JAXB und SAAJ auf
- Es gibt auch nicht JAX-WS-kompatible Frameworks, z.Bsp. Spring Web Services

Jakarta EE 8 (2019)

- Full Compatibility with Java EE 8: d.h. keine Neuerungen!

Jakarta EE 9 (2020)

- „The goal of the Jakarta EE 9 release is to deliver a set of specifications functionally similar to Jakarta EE 8 but in the new Jakarta EE 9 namespace jakarta.“
- Java SE 8 ist Basis
- Keine funktionalen Erweiterungen !
- JAX-WS ist nicht mehr Bestandteil der JEE-Plattform, aber natürlich potentiell optionale Komponente !!

JAX-WS Versionen

- JEE 8: JAX-WS (Java API for XML-Based Web Services) 2.2
  - Java Architecture for XML Binding (JAXB) 2.2
  - SOAP with Attachments API for Java (SAAJ) 1.3
  - Web Services Metadata for the Java Platform 2.1

- Jakarta EE 9: Jakarta XML Web Services 3.0, aber keine funktionalen Erweiterungen
  - Jakarta Enterprise Web Services 2.0
  - Jakarta XML Binding 3.0
  - Jakarta SOAP with Attachments 2.0
  - Jakarta Web Services Metadata 3.0

- Jakarta EE 10: Jakarta XML Web Services 4.0
  - Jakarta XML Binding 4.0
  - Jakarta SOAP with Attachments 3.0

Programmierung: Annotations Everywhere

```
@WebService  @Stateless
public class Service {

    @Inject Credentials credentials;
    @Inject @Users EntityManager userDatabase;
    ...

    private User user;

    @TransactionAttribute(REQUIRES_NEW)
    @RolesAllowed("guest")
    public void serve() {
        ...
    }
    ...
```

- oder aber Nutzung von Konfigurations-Dateien